Outer and Anti Joins in Temporal-Probabilistic Databases

Papaioannou, Katerina ; Theobald, Martin ; Böhlen, Michael

Abstract: The result of a temporal-probabilistic (TP) join with negation includes, at each time point, the probability with which a tuple of a positive relation p matches none of the tuples in a negative relation n, for a given join condition . For the computation of TP joins with negation, we introduce generalized lineage-aware temporal windows, a mechanism that binds an interval to the lineages of all the matching valid tuples of each input relation. We compute these windows in an incremental manner, and we show that pipelined computations allow for the direct integration of our approach into PostgreSQL. We thereby alleviate the prevalent redundancies in the interval computations of existing approaches, which is proven by an extensive experimental evaluation with real-world datasets.

DOI: https://doi.org/10.1109/ICDE.2019.00187

Outer and Anti Joins in Temporal-Probabilistic Databases

Katerina Papaioannou*, Martin Theobald†, Michael Böhlen*

* *Department of Computer Science – University of Zurich*
{papaioannou, boehlen}@ifi.uzh.ch

† *Faculty of Science, Technology & Communication – University of Luxembourg*
martin.theobald@uni.lu

***Abstract*—The result of a temporal-probabilistic (TP) join with negation includes, at each time point, the probability with which a tuple of a positive relation $\mathbf{p}$ matches *none* of the tuples in a negative relation $\mathbf{n}$, for a given join condition θ. For the computation of TP joins with negation, we introduce *generalized lineage-aware temporal windows*, a mechanism that binds an interval to the lineages of all the matching valid tuples of each input relation. We compute these windows in an incremental manner, and we show that pipelined computations allow for the direct integration of our approach into PostgreSQL. We thereby alleviate the prevalent redundancies in the interval computations of existing approaches, which is proven by an extensive experimental evaluation with real-world datasets.**

I. Introduction

The result of a temporal-probabilistic join with negation includes, at each time point, the probability with which a tuple of the positive relation $\mathbf{p}$ matches no tuple in the negative relation $\mathbf{n}$, for a given join condition θ. Firstly, it includes output tuples that span subintervals when only a tuple p of $\mathbf{p}$ is valid. In such cases, output intervals might be determined by starting or ending points of input tuples that are not valid during the output interval. Secondly, TP joins with negation produce outputs that indicate, at each time point, the probability of a tuple $\tilde{p}$ in $\mathbf{p}$ not matching any valid tuple in $\mathbf{n}$ because all of them are false. In this case, the lineages of these tuples are used in a *negating* form and an output interval T is determined based on the starting and ending points of $\tilde{p}$ and of the tuples of $\mathbf{n}$ that are valid over T and match θ.

Example 1: Consider a booking website (Fig. 1) that archives prediction data over time. Table $\mathbf{a}$ records data related to the locations that the clients want to visit, according to their searches. Table $\mathbf{b}$ records data regarding the availability of the hotels registered in the website, considering the busy periods in each location and the rate at which each hotel gets booked. Tuple ('Jim, WEN', a_2, [7,10), 0.8) captures that, at each day from the 7^{th} to the 10^{th} of the month, 'Jim wants to visit Wengen' with probability *0.8*. The website makes a prediction for each time point and there is no other tuple in $\mathbf{a}$ that predicts the probability of 'Jim visiting Wengen' over an interval overlapping with [7,10). In order to manage supply and demand, we determine the probability with which the client will find available accommodation at their preferred location, at each time point. The corresponding query is Q = $\mathbf{a}$ $\tilde{\bowtie}^{\text{Tp}}_{\theta}$ $\mathbf{b}$ (θ : $\mathbf{a}$.Loc = $\mathbf{b}$.Loc), i.e., a temporal-probabilistic outer join with equality on the locations.

a (wantsToVisit)

Name	*Loc*	λ	T	p
Ann	ZAK	a_1	[2,8)	0.7
Jim	WEN	a_2	[7,10)	0.8

b (hotelAvailability)

Hotel	*Loc*	λ	T	p
hotel_3	SOR	b_1	[1,4)	0.9
hotel_2	ZAK	b_2	[5,8)	0.6
hotel_1	ZAK	b_3	[4,6)	0.7

(a) Temporal-probabilistic base relations

Q = $\mathbf{a}$ $\tilde{\bowtie}^{\text{Tp}}_{\theta}$ $\mathbf{b}$, θ : $\mathbf{a}$.Loc = $\mathbf{b}$.Loc

Name	*Loc*	*Hotel*	λ	T	p
Ann	ZAK	-	a_1	[2,4)	0.70
Ann	ZAK	hotel_1	$a_1 \wedge b_3$	[4,6)	0.49
Ann	ZAK	hotel_2	$a_1 \wedge b_2$	[5,8)	0.42
Ann	ZAK	-	$a_1 \wedge \neg b_3$	[4,5)	0.21
Ann	ZAK	-	$a_1 \wedge \neg(b_3 \vee b_2)$	[5,6)	0.084
Ann	ZAK	-	$a_1 \wedge \neg b_2$	[6,8)	0.28
Jim	WEN	-	a_2	[7,10)	0.80

(b) Temporal-probabilistic tuple-based query

Fig. 1: Temporal-probabilistic database example

The answer tuple ('Ann, ZAK, hotel_1', $a_1 \wedge b_3$, [4,6), 0.49) expresses that, with probability 0.49, Ann wants to visit Zakynthos (a_1) and stay at hotel_1 in Zakynthos (b_3) during interval [4,6). It is valid over the intersection of the intervals of tuples a_1 and b_3 and it is true when both these tuples are true. Answer tuple ('Ann, ZAK, -', a_1, [2,4), 0.7) expresses that, with probability 0.7, Ann wants to visit Zakynthos (a_1) but there is no hotel available to stay there. Although the lineage and the output probability are both determined by tuple a_1, i.e., the only tuple valid during [2,4), the interval of this output tuple is influenced by the starting point of tuple b_3, a tuple not valid over [2,4). Over the interval [5,6) there is 0.084 probability that Ann wants to visit Zakynthos but finds no accommodation. According to answer tuple ('Ann, ZAK, - ', $a_1 \wedge \neg(b_3 \vee b_2)$, [5,6), 0.084), during [5,6), the output is influenced by more than a pair of input tuples. Although all tuples are valid over [5,6), this tuple is true when 'Ann visits ZAK' (a_1 is true) but also when neither hotel_1 nor hotel_2 are available during [5,6) (b_3 and b_2 are false).

TABLE I

Overlapping Windows	$\tilde{w} \in \mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta) \Longleftrightarrow \exists r \in \mathbf{r},\ s \in \mathbf{s}\ (\ \tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.F_s = s.F\ \wedge$ $\theta\ \wedge \tilde{w}.\lambda_r \equiv r.\lambda\ \wedge\ \tilde{w}.\lambda_s \equiv s.\lambda\ \wedge\ \tilde{w}.T = r.T \cap s.T\ )$
Unmatched Windows	$\tilde{w} \in \mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta) \Longleftrightarrow \tilde{w}.\lambda_s = \texttt{null}\ \wedge \tilde{w}.F_s = \texttt{null}\ \wedge$ $\forall t \in \tilde{w}.T\ (\exists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F\ \wedge \tilde{w}.\lambda_r \equiv r.\lambda)\ \wedge \tilde{w}.\lambda_s \equiv \lambda_t^{\mathbf{s},\theta_{\tilde{w}}}\ \wedge\ \lambda_t^{\mathbf{s},\theta_{\tilde{w}}} = \texttt{null}) \wedge$ $\forall t' \in \{\tilde{w}.T_s - 1, \tilde{w}.T_e\}\ (\nexists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F \wedge \tilde{w}.\lambda_r \equiv r.\lambda) \vee \tilde{w}.\lambda_s \not\equiv \lambda_{t'}^{\mathbf{s},\theta_{\tilde{w}}})$
Negating Windows	$\tilde{w} \in \mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta) \Longleftrightarrow \forall t \in \tilde{w}.T\ (\exists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.\lambda_r \equiv r.\lambda)\ \wedge$ $\tilde{w}.F_s = \texttt{null} \wedge\ \lambda_t^{\mathbf{s},\theta_{\tilde{w}}} \neq \texttt{null} \wedge \tilde{w}.\lambda_s = \lambda_t^{\mathbf{s},\theta_{\tilde{w}}}\ )\ \wedge$ $\forall t' \notin \tilde{w}.T\ (\ \nexists r \in \mathbf{r}\ (\tilde{w}.F_r = r.F\ \wedge\ \tilde{w}.\lambda_r \equiv r.\lambda)\ \vee \tilde{w}.\lambda_s \not\equiv \lambda_{t'}^{\mathbf{s},\theta_{\tilde{w}}}\ )$

Outline & Contributions.

- We introduce *generalized lineage-aware temporal windows* to produce output tuples for input pairs with different non-temporal attributes and for cases when multiple input tuples are valid. Given a θ-condition and two TP relations, we group windows into three disjoint sets: the *unmatched*, the *overlapping* and the *negating windows*. An output tuple is formed for each window using the appropriate lineage-concatenation functions.
- We introduce the algorithms LAWA_{U} and LAWA_{N} for the computation of unmatched and negating windows, respectively. Recording the lineages of the tuples valid in each input relation over an output interval and keeping them decoupled until the formation of output tuples, allows for the computation of unmatched and negating windows based on the overlapping ones. In contrast to previous works in either temporal or probabilistic databases, our approach involves no tuple replication. Instead, it allows for a pipelined calculation of the result and thus enables its smooth integration in the kernel of a DBMS.
- We conduct extensive experiments using real datasets to compare our approach for the computation of TP outer and anti joins with existing state of the art approaches. Our approach is integrated in PostgreSQL and improves the runtime for TP outer and anti joins by two orders of magnitude.

II. Generalized Windows

The use of a general θ condition in TP outer and anti joins requires pairing input tuples that include different facts and grouping multiple input tuples that are valid over an interval and satisfy θ. For this purpose, we introduce ***generalized lineage-aware temporal windows***, a mechanism with schema (F_r, F_s, T, $\lambda_{\mathbf{r}}$, $\lambda_{\mathbf{s}}$), created based on two TP relations $\mathbf{r}$ and $\mathbf{s}$, . F_r and F_s are the facts included in tuples of relations $\mathbf{r}$ and $\mathbf{s}$ over interval T, respectively. $\lambda_{\mathbf{r}}$ is the disjunction of the lineage expressions of the tuples of relation $\mathbf{r}$ that are valid over T, include F_r and satisfy θ. $\lambda_{\mathbf{s}}$ is the disjunction of the lineage expressions of the tuples of relation $\mathbf{s}$ that are valid over T and satisfy θ.

Definition 1: Let $\mathbf{r}$ and $\mathbf{s}$ be TP relations with schema (F, λ, T, p) and θ a condition between the non-temporal attributes of $\mathbf{r}$ and $\mathbf{s}$. Let $\lambda_t^{\mathbf{r},\theta}$ be the disjunction of the lineage expressions of the tuples in $\mathbf{r}$ that satisfy θ and are valid at time point t. The unmatched $\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)$, overlapping $\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)$ and negating $\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta)$ windows of $\mathbf{r}$ with respect to $\mathbf{s}$ and θ are defined according to Table I.

The *overlapping windows* $\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)$ span a maximal interval over which a tuple r of $\mathbf{r}$ overlaps with a tuple s from $\mathbf{s}$ and the predicate θ is satisfied. The *unmatched windows* $\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)$ span over the interval or a subinterval of a tuple r of $\mathbf{r}$ during which all tuples of $\mathbf{s}$ are either not valid or do not satisfy θ. The fact F_r and the lineage λ_r of an unmatched window are determined by r while F_s and λ_s are set to `null`. The negating windows $\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta)$ span intervals during which a fact is included in a tuple r of $\mathbf{r}$ as well as in multiple tuples of $\mathbf{s}$ that are valid and satisfy the θ-condition. They are suitable for producing output tuples where, for θ, all the tuples of $\mathbf{s}$ that match a tuple r of $\mathbf{r}$ including the fact F_r are *false*. Thus, the fact F_r and the lineage λ_r of the window are determined by r, F_s is set to `null` and λ_s is the disjunction of the lineages of all the tuples in $\mathbf{s}$ that match r.

Example 2: In Fig. 2, the TP relations $\mathbf{a}$ and $\mathbf{b}$ of Fig. 1 are illustrated along with the unmatched, overlapping and negating windows of $\mathbf{a}$ with respect to $\mathbf{b}$. Single lines are used for tuples and pairs of lines for windows. Different colors are used to annotate different facts: black is used for 'Ann, ZAK', red for 'John, WEN', green for 'hotel$_3$, SOR', yellow for 'hotel$_2$, ZAK', and blue for 'hotel$_1$, ZAK'. Wavy lines are used for tuples of an input relation that match no tuple of the other relation for θ. For the unmatched window w_1 = ('Ann, ZAK, `null`', [2, 4), a_1, `null`), the straight black line indicates that the fact $w_1.F_r$ = 'Ann, ZAK' and the lineage $w_1.\lambda_r = a_1$ match the corresponding attributes of tuple a_1. The dotted line indicates that fact $w_1.F_s$ is `null` and so is $w_1.\lambda_s$. At $t = 4$, a_1 is still valid whereas $\lambda_4^{\mathbf{b},\theta_{w_1}} = b_3$, which indicates that a tuple of $\mathbf{b}$ starts being valid and thus $[2, 4)$ is maximal. The window w_3 = ('Ann, ZAK', 'hotel$_1$', [4,6), a_1, b_3) is an overlapping window. For the negating window w_6 = ('Ann, ZAK', `null`, $[5, 6)$, a_1, $b_3 \vee b_2$), the black straight line in w_6 indicates that its fact F_r and its lineage λ_r correspond to the fact and lineage of a_1. The fact F_s is `null`, illustrated by a dotted line. Annotated next to this line, the λ_s equals the disjunction of the tuples b_2 and b_3 that satisfy θ over the interval $[5, 6)$. The interval $[5, 6)$ is maximal since at $t = 6$, b_3 stops being valid.

An output tuple is formed for each window using the facts (F_r, F_s) and interval T in their exact form while the output lineage is formed by combining λ_r and λ_s with the proper lineage-concatenation function. According to their semantics, each set of windows is matched with a unique function: for

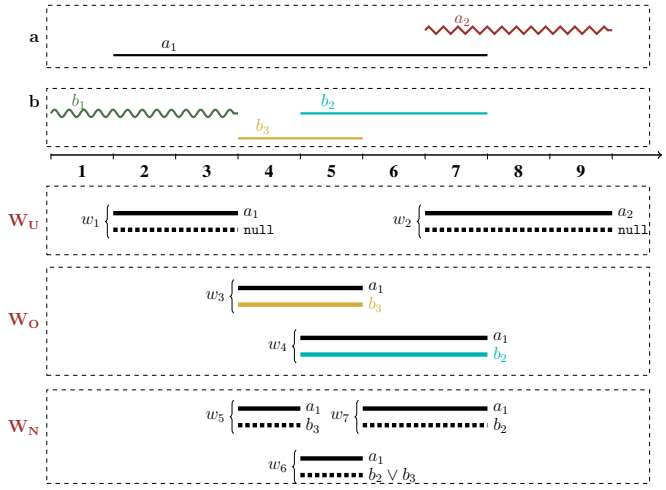


Fig. 2: All windows of $\mathbf{a}$ with respect to $\mathbf{b}$ with θ : $\mathbf{a}$.Loc = $\mathbf{b}$.Loc

overlapping windows we use the function ***and***, for *negating windows* we use ***andNot*** and for *unmatched windows* only λ_r is passed on to the output lineage. For the TP outer join in Figure 1b, the unmatched window ('Ann, ZAK', `null`, [2,4), a_1, `null`) is transformed to the output tuple ('Ann, ZAK', -, [2,4), a_1) and the negating window ('Ann, ZAK', `null`, [5,6), a_1, $b_3 \vee b_2$) to the output tuple ('Ann, ZAK', [5,6), $a_1 \wedge \neg(b_3 \vee b_2)$). In Table II, we include all the window sets required for each TP join with negation considering that $\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta) = \mathbf{W}_{\text{O}}(\mathbf{s};\mathbf{r},\theta)$.

TABLE II: TP Joins with Negation using Windows

$\mathbf{op}^{TP}$	$\mathbf{W}_{\text{U}}(\mathbf{r};\mathbf{s},\theta)$	$\mathbf{W}_{\text{N}}(\mathbf{r};\mathbf{s},\theta)$	$\mathbf{W}_{\text{O}}(\mathbf{r};\mathbf{s},\theta)$	$\mathbf{W}_{\text{U}}(\mathbf{s};\mathbf{r},\theta)$	$\mathbf{W}_{\text{N}}(\mathbf{s};\mathbf{r},\theta)$
$\mathbf{r} \triangleright \mathbf{s}$	✓	✓			
$\mathbf{r} \overline{\bowtie} \mathbf{s}$	✓	✓	✓		
$\mathbf{r} \overline{\bowtie}\!\!\!\!\bowtie \mathbf{s}$			✓	✓	✓
$\mathbf{r} \overline{\bowtie}\!\!\!\!\bowtie \mathbf{s}$	✓	✓	✓	✓	✓

III. Algorithms

In this section, we introduce two sweeping-window algorithms [1] for the computation of unmatched (LAWA_{U}) and negating (LAWA_{N}) windows. LAWA_{U} is applied on the set of overlapping windows and LAWA_{N} is applied on the windows produced by LAWA_{U}. Consequently, we avoid reduntant interval comparison and recomputing the overlapping windows multiple times, as a pipelined DBMS (like PostgreSQL) would alternatively require. Due to space constraints, we refer the reader to [2] for a more detailed description of our algorithms.

A. Overlapping Windows

Initially, we perform the conventional outer join $\mathbf{r} \overline{\bowtie}_{\theta_o \wedge \theta} \mathbf{s}$ with the overlapping predicate $\theta_o : r.T \cap s.T$ and a condition θ on the non-temporal attributes, as provided in the TP join to be computed. $\mathbf{r} \overline{\bowtie}_{\theta_o \wedge \theta} \mathbf{s}$ computes the overlapping windows of relation $\mathbf{r}$ with respect to $\mathbf{s}$, enhancing every window with the initial time-interval of the tuple of r valid over each window. It also includes all the unmatched windows where input tuples of $\mathbf{r}$ don't overlap or satisfy θ with any tuple in $\mathbf{s}$.

B. Unmatched Windows

The algorithm LAWA_{U} extends the result of $\mathbf{r} \overline{\bowtie}_{\theta_o \wedge \theta} \mathbf{s}$ with the remaining unmatched windows, i.e., the windows that span a *subinterval* of a tuple in $\mathbf{r}$ during which no tuple in $\mathbf{s}$ is valid or satisfies θ. For these unmatched windows to be created, the windows in $\mathbf{r} \overline{\bowtie}_{\theta_o \wedge \theta} \mathbf{s}$ are grouped according to the fact F_r and the interval $[T_s, T_e)$ of the tuple in $\mathbf{r}$ to which they correspond. Within each group, the tuples are sorted on the starting point of the overlapping intervals. The algorithm performs a sweep of the initial interval of each tuple r of $\mathbf{r}$. It copies the existing unmatched and the overlapping windows relating to r to the output. At the same time, given the subintervals that the overlapping windows span and the initial interval of r, it identifies the subintervals during which there is no overlap with a tuple in $\mathbf{s}$, i.e., no overlapping window, and produces the remaining unmatched windows. In Fig. 3, we illustrate the cases that LAWA_U checks for determining the ending point `windTe` of a sweeping window $[\texttt{windTs}, \texttt{windTe})$.

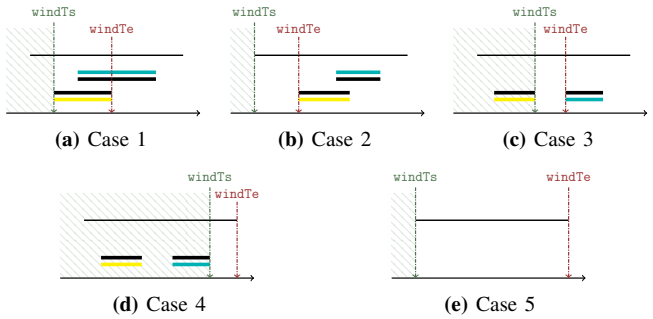


Fig. 3: LAWA_U Cases. Single line is used for the interval of the valid tuple r and pairs of lines for the windows.

C. Negating Windows

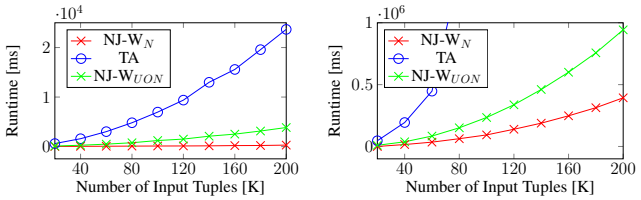
LAWA_{N} extends the result $\mathbf{W}_{\text{UO}}$ of LAWA_{U}, including all overlapping and unmatched windows with the negating windows. The windows in $\mathbf{W}_{\text{UO}}$ are ordered by the fact of $\mathbf{r}$ (F_r) as well as by their starting point. LAWA_{N} sweeps over $\mathbf{W}_{\text{UO}}$ and produces negating windows when a group of overlapping windows with the same fact F_r is encountered. A new window is created at every starting and ending point in group, i.e. every time a tuple starts or stops being valid. The intervals of the negating windows are subintervals of the group of overlapping windows. The ending points and lineages of the tuples of relation s in the overlapping windows are recorded in a priority queue. This queue facilitates determining the ending points and the lineage λ_s in the negating windows produced. The unmatched and overlapping windows in $\mathbf{W}_{\text{UO}}$ need to be also copied to the output. Such a copy and the creation of a negating window alternate. In Fig. 4, we illustrate the cases that LAWA_N checks for determining the ending point `windTe` of a sweeping window $[\texttt{windTs}, \texttt{windTe})$.

IV. Evaluation

In this section, we compare our approach for TP joins with negation (NJ) to Temporal Alignment (TA) [3], the only related

(a) Case 1 **(b)** Case 2 **(c)** Case 3

Fig. 4: LAWA$_N$ Cases. Ending point candidates are illustrated with circles (denoting the ending points in the priority queue) and crosses (denoting the upcoming starting points). In Case 1, an overlapping window is copied and in Case 3 a new group follows.

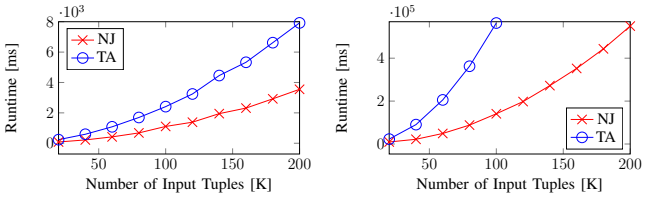


(a) Webkit **(b)** Meteo

Fig. 6: Negating Windows

approach that can be adapted for the computation of TP joins with negation. Both approaches have been implemented in the kernel of PostgreSQL 9.4.3 in C by modifying the parser, executor and optimizer. All of the following experiments were deployed on a 2xIntel(R) Xeon(R) CPU E5-24400 @2.40GHz machine with 64GB main memory. All experiments were performed in main-memory and no indexes were used.

We evaluate our algorithms using two real-world datasets. The Webkit dataset[1] [4], [5], [6] records predictions that a file remains unchanged over an interval. The Meteo Swiss dataset [2] includes predictions that a metric at a meteorological station does not vary by more that 0.1 over an interval. For the Webkit dataset, we combine tuples referring to the same file, and for Meteo tuples with measurements on the same metric but in different stations.



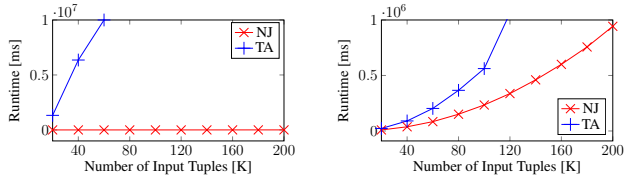
(a) Webkit **(b)** Meteo

Fig. 5: W$_{UO}$: Overlapping and Unmatched Windows

For the computation of overlapping and unmatched windows (Fig. 5), both approaches follow a similar trend since the most computationally demanding part of both is a conventional left join. NJ only executes this join once whereas TA executes it twice. As a result, NJ is two to four times faster.

In NJ, negating windows are computed by applying $LAWA_{\text{N}}$ on the set W_{UO}. In Fig. 6, we have illustrated its computatinon time for negating both including W_{UON}) and excluding (W_N) the runtime for W_{UO}. For W_{UON}, NJ computes the negating windows four to ten times faster than TA whereas, in the case of W_N, it computes them twelve to twenty times faster.

Finally, for a TP left-outer join (Fig. 7), TA's runtime is much higher than the sum of the runtimes of Fig. 5 and Fig. 6. The union combining the subresults has to remove the unmatched windows that are computed twice and when used, the θ condition of the TP join is ignored. The optimizer opts for a nested loop for $\mathbf{r} \rtimes_{\theta_o \wedge \theta} \mathbf{s}$ and this takes a huge toll on TA's runtime making NJ two orders of magnitude faster. Moreover, Meteo dataset contains a number of distinct values much smaller than its size, an analogy maintained in the subsets due to the use of the uniform distribution in their creation. As a result, the condition is not very selective and the runtime of both NJ and TA is higher than it was in the case of the webkit dataset. In all cases, the runtime of NJ outperforms TA by four to ten times.



(a) Webkit **(b)** Meteo

Fig. 7: TP Left Outer-Join

V. Conclusions

We proposed an approach for the computation of temporal-probabilistic joins with negation by introducing generalized lineage-aware temporal windows, to bind lineages and intervals and comply with the requirements of TP joins. We implemented algorithms for the pipelined computation of all sets of generalized lineage-aware temporal windows and we integrated our approach in the kernel of PostgreSQL. A thorough experimental evaluation reveals that our implementation is seamlessly integrated into the DBMS and outperforms existing approaches.

References

[1] K. Papaioannou, M. Theobald, and M. Böhlen, "Supporting set operations in temporal-probabilistic databases," in *ICDE*, 2018, pp. 1180–1191.

[2] ——, "Generalized lineage-aware temporal windows: Supporting outer and anti joins in temporal-probabilistic databases," *CoRR*, vol. abs/1902.04379, 2019. [Online]. Available: https://arxiv.org/abs/1902.04379

[3] A. Dignös, M. H. Böhlen, J. Gamper, and C. S. Jensen, "Extending the Kernel of a Relational DBMS with Comprehensive Support for Sequenced Temporal Queries," *TODS*, vol. 41, no. 4, pp. 26:1–26:46, 2016.

[4] A. Dignös, M. H. Böhlen, and J. Gamper, "Overlap interval partition join," in *SIGMOD*, 2014, pp. 1459–1470.

[5] D. Piatov, S. Helmer, and A. Dignös, "An interval join optimized for modern hardware," in *ICDE*, 2016, pp. 1098–1109.

[6] F. Cafagna and M. H. Böhlen, "Disjoint interval partitioning," *VLDB J.*, vol. 26, no. 3, pp. 447–466, 2017.

[1]The WebKit Open Source Project: http://www.webkit.org (2012)

[2]Federal Office of Meteorology and Climatology: http://www.meteoswiss.ch (2016)